Objectives

- Search comparison
- Two-dimensional lists

1

Review

- What are the two types of search we discussed?
  - How do they work?
  - How do they compare?
  - What are the tradeoffs between using linear search and binary search?

2

Review: Search Using `in` Review

- Iterates through a list, checking if the element is found
- Known as ***linear search***
- **Implementation:**

```
def linearSearch(searchlist, key):
    for elem in searchlist:
        if elem == key:
            return True
    return False
```

value	8	5	3	7
pos	0	1	2	3

3

Alternative: Like `index` method

- Iterates through positions in a list, checking if the element is found
- Still known as ***linear search***
- **Implementation:**

```
def linearSearch(searchlist, key):
    for pos in len(range(searchlist)):
        if searchlist[pos] == key:
            return pos
    return -1
```


4

Review: Linear Search

- **Overview**: Iterates through a list, checking if the element is found
- **Benefits:**
  - Works on *any* list
- **Drawbacks**:
  - **Slow**, on average: needs to check each element of list if the element is not in the list

5

Review: Binary Search: Eliminate Half the Possibilities

- Repeat until find value (or looked through all values)
  - Guess middle *value* of possibilities
    - (not middle *position*)
  - If match, found!
  - Otherwise, find out too high or too low
  - Modify your possibilities
    - Eliminate the possibilities from your number and higher/lower, as appropriate
- Known as **Binary Search**

6

Binary Search Implementation

```
def search(searchlist, key):
   low=0
   high = len(searchlist)-1
   while low <= high :
      mid = (low+high)//2
      if searchlist[mid] == key:
         return mid    # return True
      elif key > searchlist[mid]:
         low = mid+1
      else:
         high = mid-1
   return -1    # return False
```

If you just want to know if it's in the list

Binary Search

- Example of a ***Divide and Conquer*** algorithm
  - Break into smaller pieces that you can solve
- Benefits:
  - Faster to find elements (especially with larger lists)
- Drawbacks:
  - List **must** be sorted before searching
    - Takes time to sort
  - Requires that data can be compared
    - `__lt__, __eq__` methods implemented by the class (or another solution)

More on this tomorrow

Key Questions in Computer Science

- How can we efficiently organize data?
- How can we efficiently search for data, given various constraints?
  - Example: data may or may not be sortable
- What are the tradeoffs?

Empirical Study of Search Techniques

Goal: Determine which technique is better under various circumstances

- How long does it take to find various keys?
  - **Measure** by the number of comparisons
  - Vary the size of the list and the keys
  - What are good tests for the lists and the keys?

`search_compare.py`

Empirical Study of Search Techniques

- Analyzing Results …
  - By how much did the number of comparisons for *linear search* vary?
  - By how much did the number of comparisons for *binary search* vary?
- What conclusions can you draw from these results?

`search_compare.py`

11

Search Strategies Summary

- Which search strategy should I use under the following circumstances?
  - I have a short list
  - I have a long list
  - I have a long sorted list

12

Search Strategies Summary

- Which search strategy should I use under the following circumstances?
  - I have a short list
    - How short? How many searches? Linear (`in`)
  - I have a long list
    - Linear (`in`) - because don't know if in order, comparable
    - Alternatively, may want to sort the list and *then* perform binary search, if sorting first won't be more effort than just sorting.
  - I have a long sorted list
    - Binary

2D LISTS

Lists

- We've used lists that contain
  - Integers
  - Strings
  - Cards (Deck class)
  - Persons (your Person class)
- We discussed that lists can contain multiple types of objects within the same list
  - Wheel of Fortune: ["Bankrupt", 250, 350, …]
- Lists can contain *any* **type** of object
  - Even **LISTS**!

Review of Regular (1D) Lists

```
onedlist = [ 7, -1, 23 ]
```

Elements in the list

- How do we find the number of elements in the list?
- How can we find the value of the third element in the list?

Review of Regular (1D) Lists

```
onedlist = [ 7, -1, 23 ]
```

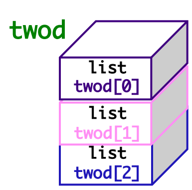
Elements in the list

- `len(onedlist)` is 3
- `onedlist[2]` is 23

17

A List of Lists: 2-Dimensional List

```
            twod[0]   twod[1]     twod[2]
twod = [ [1,2,3,4], [5,6], [7,8,9,10,11] ]
```

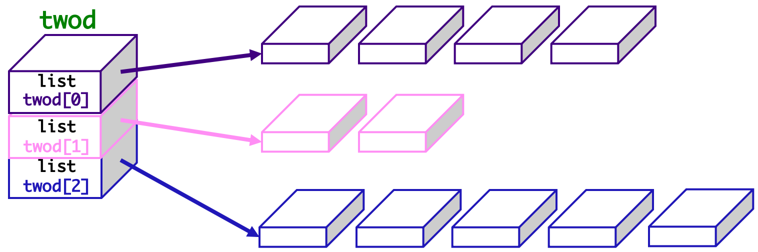


1st dimension

18

A List of Lists: 2-Dimensional list

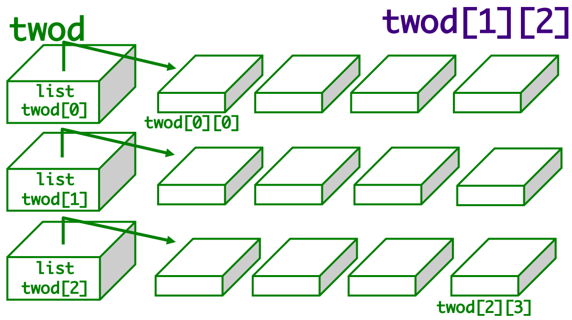
```
twod = [ [1,2,3,4], [5,6], [7,8,9,10,11] ]
```



- "Rows" within 2-dimensional list do **not** need to be the same length
- However, it's often easier if they're the same length!
  - We'll focus on "**rectangular**" 2D lists

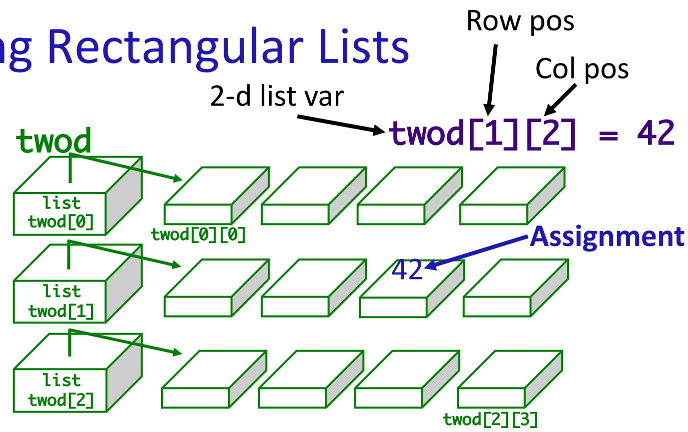
Handling Rectangular Lists

```
twod[1][2] = 42
```



- What does each component of `twod[1][2]` mean?
- How can we programmatically determine the number of rows in `twod`? The number of columns in a given row?

Handling Rectangular Lists



- How can we programmatically determine the number of rows in `twod`?
  - ➢ `rows = len(twod)`
- The number of columns in a given row?
  - ➢ `cols = len(twod[whichRow])`

21

2D List Practice

twod Before

	col 0	col 1	col 2	col 3
row 0 →	1	2	3	4
row 1 →	5	6	7	8
row 2 →	9	10	11	12

Starting with the 2D list **twod** shown here, what are the values in **twod** after running this code?

```
def mystery(twod):
    """ 'run' this on twod, at right """
    for row in range( len(twod) ):
        for col in range( len(twod[row]) ):
            if row == col:
                twod[row][col] = 42
            else:
                twod[row][col] += 1
```

twod After

mystery.py

22

2D List Practice

Starting with the 2D list **twod** shown here, what are the values in **twod** after running this code?

twod Before

	col 0	col 1	col 2	col 3
row 0	1	2	3	4
row 1	5	6	7	8
row 2	9	10	11	12

```
def mystery(twod):
    """ 'run' this on twod, at right """
    for row in range( len(twod) ):
        for col in range( len(twod[row]) ):
            if row == col:
                twod[row][col] = 42
            else:
                twod[row][col] += 1
```

twod After

42	3	4	5
6	42	8	9
10	11	42	13

mystery.py

Creating a 2D List

`twod = [ ]`

- Create a row of the list, e.g.,
  `row = [1, 2, 3, 4]` or `row = list(range(1,5))`
  or `row = [0] * 4` or …
- Then append that row to the list
  ```
  twod.append( row )
  print(twod)
  ```
  - [[1, 2, 3, 4]]
- Repeat
  ```
  row = list(range(1,5))
  twod.append( row )
  print(twod)
  ```
  - [[1, 2, 3, 4], [1, 2, 3, 4]]

Generalize Creating a 2D List

- Create a function that returns a 2D list with width ***cols*** and height ***rows***
  - Initialize each element in (sub) list to 0

Generalize Creating a 2D List

- Create a function that returns a 2D list with width ***cols*** and height ***rows***
  - Initialize each element in (sub) list to 0

```
def create2DList(rows, cols):
   twodlist = [ ]
   # for each row
   for rowPos in range( rows ):
      row = [ ]
      # for each column, in each row
      for colPos in range( cols ):
          row.append(0)
      twodlist.append(row)
   return twodlist
```

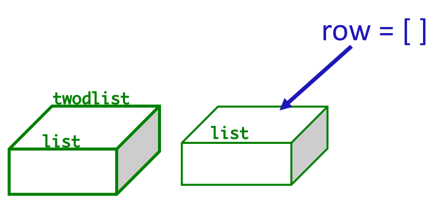
Generalize Creating a 2D List

- Create a function that returns a 2D list with width ***cols*** and height ***rows***
  - Initialize each element in (sub) list to 0

```
def create2DList(rows, cols):
   twodlist = [ ]
   # for each row
   for rowPos in range( rows ):
      row = [ ]
      # for each column, in each row
      for colPos in range( cols ):
         row.append(0)
      twodlist.append(row)
   return twodlist
```

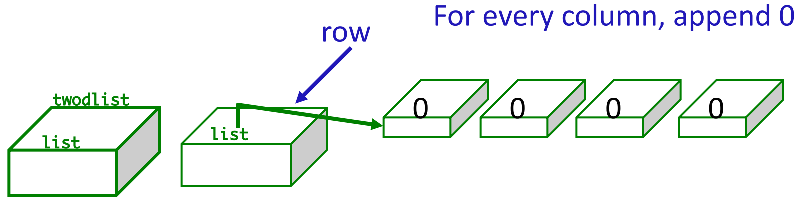
Flexibility in what you put into the list

Example: Creating 2D List – 3 rows, 4 cols



Example: Creating 2D List – 3 rows, 4 cols

For every column, append 0

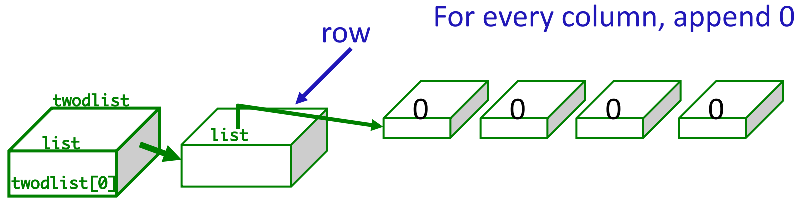


Apr 5, 2021 Sprenkle - CSCI111 29

29

Example: Creating 2D List – 3 rows, 4 cols

For every column, append 0

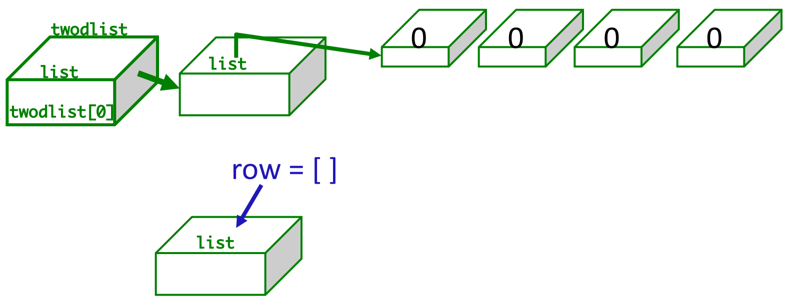


Append row to `twodlist`

Apr 5, 2021 Sprenkle - CSCI111 30

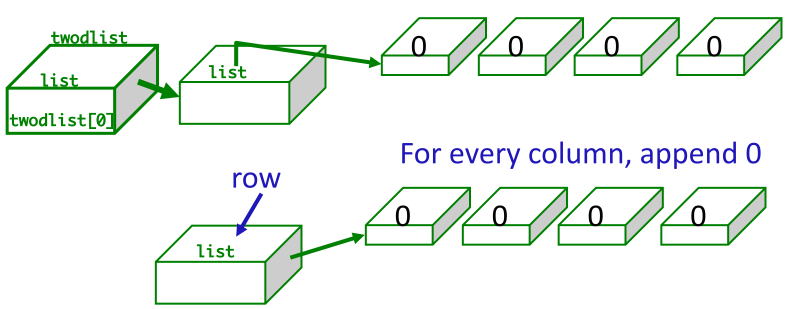
30

Example: Creating 2D List – 3 rows, 4 cols



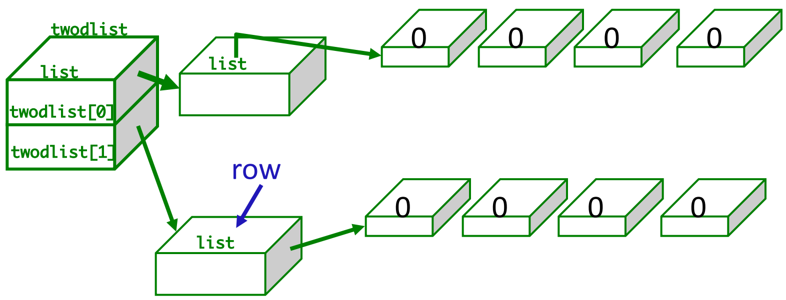
31

Example: Creating 2D List – 3 rows, 4 cols



32

Example: Creating 2D List – 3 rows, 4 cols

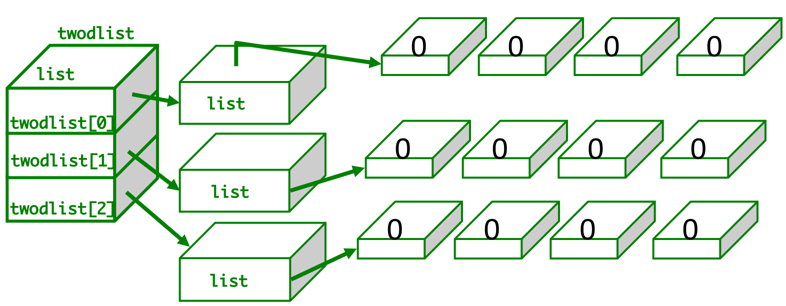


Append `row` to `twodlist`

Apr 5, 2021 Sprenkle - CSCI111 33

33

Example: Creating 2D List – 3 rows, 4 cols



Apr 5, 2021 Sprenkle - CSCI111 34

34

Generalize Creating a 2D List

- Create a function that returns a 2D list with width ***cols*** and height ***rows***
  - Initialize each element in (sub) list to 0

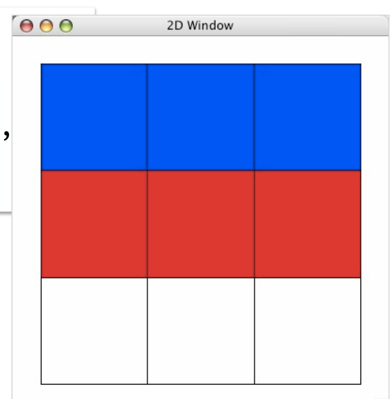
```
def create2DList(rows, cols):
   twodlist = [ ]
   # for each row
   for rowPos in range( rows ):
      row = [ ]
      # for each column, in each row
      for colPos in range( cols ):
         row.append(0)
      twodlist.append(row)
   return twodlist
```

Flexibility in what you put into the list

Graphical Representation of 2D Lists

- Module: `csplot`
- Allows you to visualize your 2D list
  - Numbers are represented by different colors

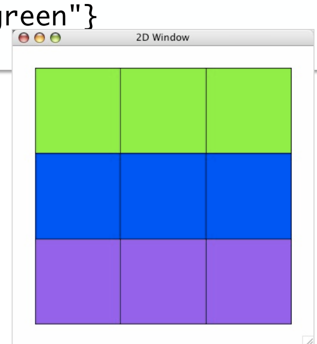
```
import csplot
…
# create 2D list…
twodlist=[ [0,0,0], [1,1,1], [2,2,
# display list graphically
csplot.show(twodlist)
```



Graphical Representation of 2D Lists

- Can assign colors to numbers

```
import csplot
…
# create 2D list…
twodlist=[ [0,0,0], [1,1,1], [2,2,2] ]
# create optional dictionary of numbers to their color rep
numToColor={0:"purple", 1:"blue", 2:"green"}
csplot.show(twodlist, numToColor)
```

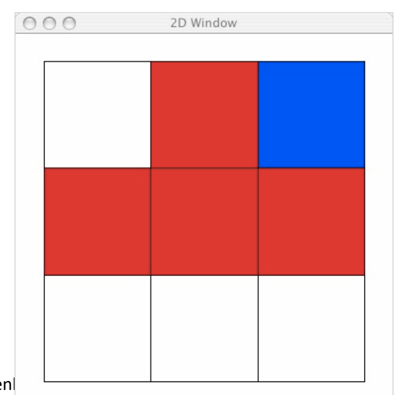


37

Graphical Representation of 2D Lists

```
matrix = [[0,0,0], [1,1,1], [0,1,2]]
```

What values map to which colors by default?



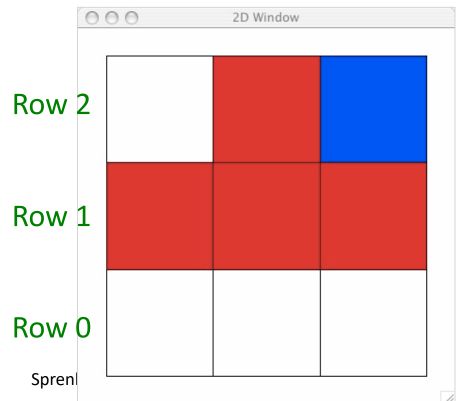
38

Graphical Representation of 2D Lists

- Note that representation of rows is backwards from how we've been visualizing

```
matrix = [[0,0,0], [1,1,1], [0,1,2]]
```

What values map to which colors by default?



39

Game Board for Connect Four

- 6 rows, 7 columns board
- Players alternate dropping red/black checker into slot/column
- Player wins when have four checkers in a row vertically, horizontally, or diagonally

How do we represent the board as a 2D list, using a graphical representation?

40

Representing Connect Four Game Board

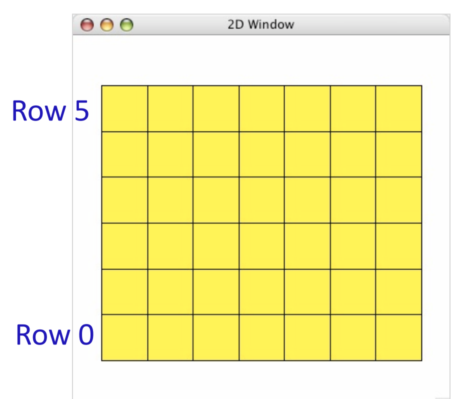
- Using a 2D list

Number	Meaning	Color
0	Free	Yellow
1	Player 1	Red
2	Player 2	Black

Representing Connect Four Game Board

- Using a 2D list

Number	Meaning	Color
0	Free	Yellow
1	Player 1	Red
2	Player 2	Black



ConnectFour Class

- What is the data associated with the class?
- What methods should we implement?

Apr 5, 2021 Sprenkle - CSCI111 43

43

ConnectFour Class

- Data
  - Board + constants
    - 6 rows, 7 columns, all FREE to start
- Methods
  - Constructor
  - Display the board
  - Play the game
  - Get input/move from user
  - Check if valid move
  - Make move
  - Check if win

Apr 5, 2021 Sprenkle - CSCI111 44

44

ConnectFour Constants

```
class ConnectFour:
    """ Class representing the game Connect Four. """

    # Represent different values on the board
    FREE = 0
    PLAYER1 = 1
    PLAYER2 = 2

    # Represent the dimensions of the board
    ROWS = 6
    COLS = 7
```

To reference constants, use `ConnectFour.CONSTANT`

ConnectFour Class

- Play the game method implementation
  - Repeat:
    - Get input/mov
    - Check if valid m
    - Make move
    - Display board
    - Check if win
    - Change player

```
def play(self):
    won = False
    player = ConnectFour.PLAYER1

    while not won:
        print("Player {:d}'s move".format(player))
        if player == ConnectFour.PLAYER1:
            col = self._userMakeMove()
        else: # computer is player 2
            # pause because otherwise move happens too
            # quickly and looks like an error
            sleep(.75)
            col = self._computerMakeMove()

        row = self.makeMove(player, col)
        self.showBoard()
        won = self._isWon(row, col)

        # alternate players
        player = player % 2 + 1
```

Connect Four (C4): Making moves

- User clicks on a column
  - "Checker" is filled in at that column

```
# gets the column where user clicked
col = csplot.sqinput()
```

```
    def _userMakeMove(self):
        """Allow the user to pick a column."""
        col = csplot.sqinput()
        validMove = self._isValidMove(col)
        while not validMove:
            print("NOT A VALID MOVE.")
            print("PLEASE SELECT AGAIN.")
            print()
            col = csplot.sqinput()
            validMove = self._isValidMove(col)
        return col
```


Problem: C4 - Valid move?

- Need to enforce valid moves
  - In physical game, run out of spaces for checkers if not a valid move
- How can we determine if a move is valid?
  - How do we know when a move is ***not*** valid?

Problem: C4 - Valid move?

- Solution: check the "top" spot
  - If the spot is FREE, then it's a valid move

Problem: C4 - Making a Move

- The player clicks on a column, meaning that's where the player wants to put a checker
- How do we update the board?

Looking Ahead

- Lab 11 – Tomorrow
  - Pre lab: review nested lists, classes
  - Review implementation of binary search
- Broader Issue: Facebook – Friday

Exam 2 Results

	A	B	C	Total
Average	87.29	78.41	84.21	89.16
Median	89.20	77.27	89.47	93.75

- Common issues
  - Identifying data types (int, str, dictionary, list)
  - Tracing functions, describing what they do
    - Formal, actual parameters
  - What code outputs
  - Complicating code to solve problem
    - Ex: can use `in` to check if a key is in a dictionary